JOURNAL OF COMPUTER AND SYSTEM SCIENCES 6, 1–8 (1971)

Computation of Powers of Multivariate Polynomials over the Integers

LEE E. HEINDEL*

Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

Received February 12, 1971

Theoretical computing time analyses of both the iterative multiplication and binary expansion algorithms for computing P^n for $P \in I[x_1, ..., x_r]$ show the iterative multiplication algorithm to be more efficient as r, the number of variables, increases.

1. INTRODUCTION

In [4], Knuth discusses the problem of minimizing the computing time required to compute x^n for x belonging to any algebraic system with associative multiplication and a multiplicative identity. The general approach taken is to develop algorithms which minimize the number of multiplications involved in the computation. This approach leads to a minimum computation time if one adds the additional assumption of a constant bound on the time to multiply any two elements of the system. The assumption of a constant bound on the multiplication time is reasonable, for instance, when the algebraic system is the integers and fixed-precision multiplication is employed on a digital computer. However, as we shall see, the assumption of a constant bound on the multiplication time for multiplying variable-precision integers (which can be considered as an abstraction of normal paper and pencil multiplication) is unrealistic.

Knuth analyzes two algorithms in some detail—the iterative multiplication algorithm and the binary expansion algorithm. The binary expansion algorithm is shown to require fewer multiplications and hence less computing time. Other methods of less generality which involve fewer multiplications than the binary expansion method are also discussed.

In this paper we extend Knuth's analysis of the iterative multiplication and binary expansion algorithms to computing powers of polynomials belonging to $I[x_1, ..., x_r]$, the set of all polynomials in r variables over the integers, using either fixed- or variable-precision integer arithmetic. It is shown that the iterative multiplication algorithm

* Current address: Bell Telephone Laboratories, Inc., Holmdel, NJ 07733.

571/6/1-1

minimizes the number of coefficient multiplications required and hence minimizes the computation time.

The iterative multiplication algorithm is shown to be faster for dense polynomials—i.e., ones with mostly nonzero coefficients—by a factor of n^{r-1} when fixed-precision arithmetic is employed and n^r when variable-precision arithmetic is employed to compute the n-th power of a polynomial with r variables, $r > 1$.

2. Some Preliminaries

In this section we derive upper bounds on the computing time to compute $P \cdot Q$ for $P, Q \in I[x_1, \ldots, x_r]$ using both fixed- and variable-precision integer arithmetic. We begin by presenting the definition of *O-notation.*

Definition 2.1. Let f and g be two real-valued functions defined on some set S. Then $f = O(g)$ if there exists a positive real number c such that $|f(x)| \leqslant c \cdot |g(x)|$ for all $x \in S$. Notice that the set S may be a set of n-tuples.

We shall now look at the time to multiply polynomials using fixed-precision arithmetic. If we bound the maximum size that any integer involved in a computation can be (i.e., use fixed-precision arithmetic), then we can find some positive constants, c_1 and c_2, such that the time to add any two integers is bounded by c_1 and the time to multiply any two integers is bounded by c_2. This is the case if single-precision integer arithmetic is used on a digital computer provided no integers larger than single-precision occur. Hence we have the following theorem:

Theorem 2.1. *Let $P, Q \in I[x_1, \ldots, x_r]$, let m_i be the degree of P in x_i, and let n_i be the degree of Q in x_i. Then an upper bound on the computing time to compute $P \cdot Q$ using fixed-precision integer arithmetic is $O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r)$, where $\lambda_i = m_i + 1$ and $\gamma_i = n_i + 1$.*

Proof. The multiplication of P times Q involves computing the product of each coefficient of P times each coefficient of Q and then summing coefficients of like terms. The polynomial P has at most $(m_1 + 1) \cdots (m_r + 1) = \lambda_1 \cdots \lambda_r$ terms and likewise Q has at most $\gamma_1 \cdots \gamma_r$ terms. Hence there are at most $\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r$ coefficient multiplications involved. If we assume c_2 bounds the time to multiply any two coefficients, then the time to compute all the products is $O(c_2 \lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r) = O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r)$.

There are at most $\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r$ additions required to combine the coefficients of like terms. If we assume c_1 bounds the time to add any two coefficients, we get a time to do all the additions of $O(c_1 \lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r) = O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r)$. And hence $O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r)$ is an upper bound on the computing time to compute $P \cdot Q$ using fixed-precision arithmetic.

In most variable-precision arithmetic systems implemented on digital computers [2] it can be shown [3] that the upper bound to add any two integers bounded in magnitude by d is $O[(\ln d)]$ and to multiply an integer bounded in magnitude by d times an integer bounded in magnitude by e is $O[(\ln d)(\ln e)]$. These times correspond to the intuitive notions that the time to add two integers is proportional to their length and the time to multiply them is proportional to the product of their lengths.

Knuth reports in [5] that Schönhage and Strassen have shown that it is possible to multiply two integers bounded in magnitude by d in a time proportional to $(\ln d)(\ln \ln d)(\ln \ln \ln d)$. However, their method is impractical except for very large integers due to the overhead involved in the computation. Hence in our analyses we shall use the multiplication time given in the preceding paragraph.

We now present the definition of the *norm* of a polynomial.

DEFINITION 2.2. Let $P \in I[x_1, \ldots, x_r]$. Then define $\operatorname{norm}(P)$ to be the sum of the absolute values of the coefficients of P.

The norm of a polynomial is easily seen to satisfy the following three properties: (1) $\operatorname{norm}(P + Q) \leqslant \operatorname{norm}(P) + \operatorname{norm}(Q)$, (2) $\operatorname{norm}(P \cdot Q) \leqslant \operatorname{norm}(P) \cdot \operatorname{norm}(Q)$, and (3) if a_i is any coefficient of P, then $| a_i | \leqslant \operatorname{norm}(P)$.

THEOREM 2.2. *Let $P \in I[x_1, \ldots, x_r]$, $\operatorname{norm}(P) = d$, and let m_i be the degree of P in x_i. Let $Q \in I[x_1, \ldots, x_r]$, $\operatorname{norm}(Q) = e$, and let n_i be the degree of Q in x_i. Then an upper bound on the computing time to compute $P \cdot Q$ using variable-precision arithmetic is $O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r (\ln d)(\ln e))$, where $\lambda_i = m_i + 1$ and $\gamma_i = n_i + 1$.*

Proof. As in the proof of Theorem 2.1 the number of coefficient multiplications and additions is $O(\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r)$. Since any coefficient of P is at most equal in magnitude to d and any coefficient of Q is at most equal in magnitude to e, each multiplication takes at most $O[(\ln d)(\ln e)]$ and all of them require $O[\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r (\ln d)(\ln e)]$. Since by the second property of norms, $\operatorname{norm}(P \cdot Q) \leqslant d \cdot e$, it follows that each addition involves adding two integers bounded by $d \cdot e$. Hence each addition requires $O[(\ln de)]$ and all the additions require $O[\lambda_1 \cdots \lambda_r \gamma_1 \cdots \gamma_r (\ln de)]$. And thus the time to do the multiplications bounds the time to do the additions.

3. THE ALGORITHMS AND THEIR ANALYSES

By the iterative multiplication algorithm for computing P^n we mean simply to compute $P^2 = P \cdot P$, $P^3 = P^2 \cdot P, \ldots, P^n = P^{n-1} \cdot P$. We now analyze the computing time of the iterative multiplication algorithm using fixed- and variable-precision arithmetic.

THEOREM 3.1. *Let $P \in I[x_1, \ldots, x_r]$ be a nonzero polynomial and let m_i be the degree of P in x_i. Then an upper bound on the computing time to compute P^n, $n > 0$, as a function of the power, the number of variables, and the degree of the variables, using the iterative multiplication algorithm and fixed-precision arithmetic is $O(n^{r+1}\lambda_1^2 \cdots \lambda_r^2)$, where $\lambda_i = m_i + 1$.*

Proof. If m_i is the degree of P in x_i, then the degree of P^k in x_i is $k \cdot m_i$. Hence by Theorem 2.1, the computing time to compute $P^{k-1} \cdot P$ is

$$\begin{aligned} &O\{[(k-1)m_1 + 1] \cdots [(k-1)m_r + 1](m_1 + 1) \cdots (m_r + 1)\} \\ &\quad = O[(k-1)(m_1 + 1) \cdots (k-1)(m_r + 1)(m_1 + 1) \cdots (m_r + 1)] \\ &\quad = O[(k-1)^r \lambda_1^2 \cdots \lambda_r^2]. \end{aligned}$$

Thus the time to compute all the required products is

$$O\left(\sum_{k=2}^{n} (k-1)^r \lambda_1^2 \cdots \lambda_r^2\right) = O(n^{r+1}\lambda_1^2 \cdots \lambda_r^2).$$

THEOREM 3.2. *Let $P \in I[x_1, \ldots, x_r]$ be a nonzero polynomial, let m_i be the degree of P in x_i, and let $d = \text{norm}(P)$. Then an upper bound on the computing time to compute P^n, $n > 0$, as a function of the power, the number of variables, the degrees of the variables, and the norm of the polynomial, using the iterative multiplication algorithm and variable-precision arithmetic is $O[n^{r+2}\lambda_1^2 \cdots \lambda_r^2 (\ln d)^2]$, where $\lambda_i = m_i + 1$.*

Proof. By the second property of norms, if $\text{norm}(P) = d$, then $\text{norm}(P^k) \leqslant d^k$. Hence by Theorem 2.2 the computing time to compute $P^{k-1} \cdot P$ is

$$\begin{aligned} &O\{[(k-1)m_1 + 1] \cdots [(k-1)m_r + 1](m_1 + 1) \cdots (m_r + 1)(\ln d^{k-1})(\ln d)\} \\ &\quad = O[(k-1)(m_1 + 1) \cdots (k-1)(m_r + 1)(m_1 + 1) \cdots (m_r + 1)(k-1)(\ln d)^2] \\ &\quad = O[(k-1)^{r+1}\lambda_1^2 \cdots \lambda_r^2 (\ln d)^2]. \end{aligned}$$

Thus the time to compute all the required products is

$$O\left(\sum_{k=2}^{n} (k-1)^{r+1}\lambda_1^2 \cdots \lambda_r^2 (\ln d)^2\right) = O[n^{r+2}\lambda_1^2 \cdots \lambda_r^2 (\ln d)^2].$$

In order to facilitate analyzing the binary expansion algorithm for computing P^n, we now present the algorithm in detail.

Binary Expansion Algorithm

Input: P, a nonzero polynomial belonging to $I[x_1, \ldots, x_r]$; and n, a positive integer.

Output: $X = P^n$

(1) $X \leftarrow 1$; $Z \leftarrow P$.

(2) $q \leftarrow [n/2]$; $r \leftarrow n - 2q$; $n \leftarrow q$; if $r = 0$, go to (4).

(3) $X \leftarrow X \cdot Z$.

(4) If $n = 0$, return; $Z \leftarrow Z \cdot Z$; go to (2).

That the binary expansion algorithm correctly computes P^n is a result of considering n as a binary number and observing the relationship between multiplication of powers and addition of exponents.

THEOREM 3.3. *Let $P \in I[x_1, \ldots, x_r]$ be a nonzero polynomial and let m_i be the degree of P in x_i. Then an upper bound on the computing time to compute P^n, $n > 0$, as a function of the power, the number of variables, and the degrees of the variables, using the binary expansion algorithm and fixed-precision arithmetic is $O(n^{2r}\lambda_1^2 \cdots \lambda_r^2)$, where $\lambda_i = m_i + 1$.*

Proof. In order to bound the computing time of the binary expansion algorithm it suffices to consider the time required for all computations of $X \leftarrow X \cdot Z$ in step (3), and $Z \leftarrow Z \cdot Z$ in step (4). Let us first look at computing $Z \leftarrow Z \cdot Z$ in step (4). We see that on successive executions of step (4), Z is computed as

$$Z = P \cdot P, \qquad Z = P^2 \cdot P^2, \ldots, Z = P^{2^{(\log_2 n)-1}} \cdot P^{2^{(\log_2 n)-1}}.$$

Hence the k-th execution of step (4) requires computing $P^{2^{k-1}} \cdot P^{2^{k-1}}$. If m_i is the degree of P in x_i, then $P^{2^{k-1}}$ is of degree $2^{k-1} \cdot m_i$ in x_i. Hence, the computation time for the k-th execution of step (4) is

$$O[(2^{r(k-1)}\lambda_1 \cdots \lambda_r)^2] = O[2^{2r(k-1)}\lambda_1^2 \cdots \lambda_r^2],$$

and thus the time for all executions of step (4) is

$$O\left(\sum_{k=1}^{[\log_2 n]} 2^{2r(k-1)}\lambda_1^2 \cdots \lambda_r^2\right) = O(n^{2r}\lambda_1^2 \cdots \lambda_r^2).$$

Let us now look at the time to compute $X \leftarrow X \cdot Z$ in step (3). Step (3) is executed once for each 1 in the binary representation of n and hence can be executed at most one time more than step (4). Since X is always a smaller power of P than Z is, it is easy to see that the time for every execution of step (3) except the last is bounded by the time of the corresponding execution of step (4). The last execution of step (3) can take at most

$$O(2^{r[(\log_2 n)^{-1}]+r(\log_2 n)}\lambda_1^2 \cdots \lambda_r^2) = O(n^{2r}\lambda_1^2 \cdots \lambda_r^2).$$

Thus $O(n^{2r}\lambda_1^2 \cdots \lambda_r^2)$ is an upper bound on the computing time of the binary expansion algorithm when fixed-precision arithmetic is employed.

THEOREM 3.4. *Let $P \in I[x_1, \ldots, x_r]$ be a nonzero polynomial, let m_i be the degree of P in x_i, and let $d = \text{norm}(P)$. Then an upper bound on the computing time to compute P^n, $n > 0$, as a function of the power, the number of variables, the degrees of the variables, and the norm of the polynomial, using the binary expansion algorithm and variable-precision arithmetic is*

$$O(n^{2r+2}\lambda_1^2 \cdots \lambda_r^2(\ln d)^2), \qquad \textit{where} \quad \lambda_i = m_i + 1.$$

Proof. As in the proof of Theorem 3.3, it is only necessary to consider the computing time for step (4). Since $\text{norm}(P^{2^{k-1}}) \leqslant d^{2^{k-1}}$, the computing time for the k-th execution of step (4) is at most:

$$O\{[(2^{r(k-1)}\lambda_1 \cdots \lambda_r(\ln d^{2^{k-1}})]^2\} = O[2^{2(r+1)(k-1)}\lambda_1^2 \cdots \lambda_r^2(\ln d)^2].$$

Thus the time for all executions of step (4) is

$$O\left(\sum_{k=1}^{(\log_2 n)} 2^{2(r+1)(k-1)}\lambda_1^2 \cdots \lambda_r^2(\ln d)^2\right) = O[n^{2r+2}\lambda_1^2 \cdots \lambda_r^2(\ln d)^2].$$

4. TWO EXAMPLES

In order to lend further credence to the results established in the preceding section we derive exact bounds on the number of multiplications used by both algorithms in two special cases, one as a function of the number of variables, and the other as a function of the power.

Consider computing the fourth power of the polynomial

$$\sum_{e_1=0}^{2}\sum_{e_2=0}^{2}\cdots\sum_{e_r=0}^{2} 1 \cdot x_1^{e_1}x_2^{e_2} \cdots x_r^{e_r}.$$

By examining the proofs of Theorem 3.1 and 3.3 it is easy to see that the exact number of multiplications used by the iterative multiplication algorithm is $3^r(3^r + 5^r + 7^r)$ and by the binary expansion algorithms is $3^{2r} + 5^{2r}$, where r is the number of variables. In Table 1, we have tabulated the values of the above functions for $r = 1, \ldots, 6$.

TABLE I

Number of Multiplications Using Iterative and Binary Expansion Algorithms as a Function of r, the Number of Variables

r	Number of multiplications using iterative	Number of multiplications using binary expansion
1	45	34
2	747	706
3	13365	16354
4	251667	397186
5	4902525	9824674
6	97688187	244672066

As a second example, let us consider computing the n-th power of the polynomial

$$\sum_{e_1=0}^{2}\sum_{e_2=0}^{2}\sum_{e_3=0}^{2}\sum_{e_3=0}^{2} 1 \cdot x_1^{e_1} \cdot x_2^{e_2} \cdot x_3^{e_3} \cdot x_4^{e_4}.$$

Again, by examining the proofs of Theorems 3.1 and 3.3 it is easy to see that the exact number of multiplications used by the iterative multiplication algorithm is $81 \cdot \sum_{k=2}^{n} (2k - 1)^4$ and by the binary expansion algorithm is $\sum_{k=1}^{\log_2 n} (2^k + 1)^8$, when n is a power of two. In Table II, we have tabulated the values of the above functions for $n = 2, 4, 8$.

TABLE II

Number of Multiplications Using Iterative and Binary Expansion Algorithms as a Function of n, the Power

n	Number of multiplications using iterative	Number of multiplications using binary expansion
2	6561	6561
4	251667	397186
8	8383095	43443907

5. Conclusion

In conclusion it is seen that, counter to intuition, the upper bound on the computing time of the binary expansion algorithm is much greater than the upper bound on the computing time of the iterative multiplication algorithm whether fixed- or variable-precision arithmetic is employed.

It should also be noted that if fixed-precision real arithmetic were used instead of fixed-precision integer arithmetic, the upper bounds would be the same as in Theorems 3.1 and 3.3.

ACKNOWLEDGMENTS

I would like to acknowledge the interest and suggestions of Professor G. E. Collins and Dr. M. R. Garey.

REFERENCES

1. G. E. COLLINS, "The SAC-1 Polynomial System," Technical Report No. 2, University of Wisconsin Computing Center, 1968.
2. G. E. COLLINS AND J. R. PINKERT, "The Revised SAC-1 Integer Arithmetic System," Technical Report No. 9, University of Wisconsin Computing Center, 1968.
3. G. E. COLLINS, Computing time analysis of some arithmetic algorithms, *in* "Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation" (R. G. Tobey, Ed.), IBM Federal Systems Center, Gaithersburg, MD, 1969.
4. D. E. KNUTH, "The Art of Computer Programming: 2. Seminumerical Algorithms," Addison-Wesley, Reading, MA, 1969.
5. D. E. KNUTH, "The Art of Computer Programming—Errata and Addenda," Computer Science Technical Report 71-194, Stanford University, 1971.